IBM Spectrum Scale: Performance Update

John Lewars

Spectrum Scale Performance
(v1.2.1)

Disclaimer

- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
- IBM reserves the right to change product specifications and offerings at any time without notice. This publication could include technical inaccuracies or typographical errors. References herein to IBM products and services do not imply that IBM intends to make them available in all countries.

Outline

- File create performance improvements in 5.0.3
- mmap Performance Improvements in 5.0.4.3
- Support for ATOMIC_OPEN coming in a future release
- Performance improvements for file create in a shared directory (on 4096 nodes) (summary of improvements from 5.0.3 +)

File Create Performance Scaling Improvements in 5.0.3

IBM Spectrum Scale

Description

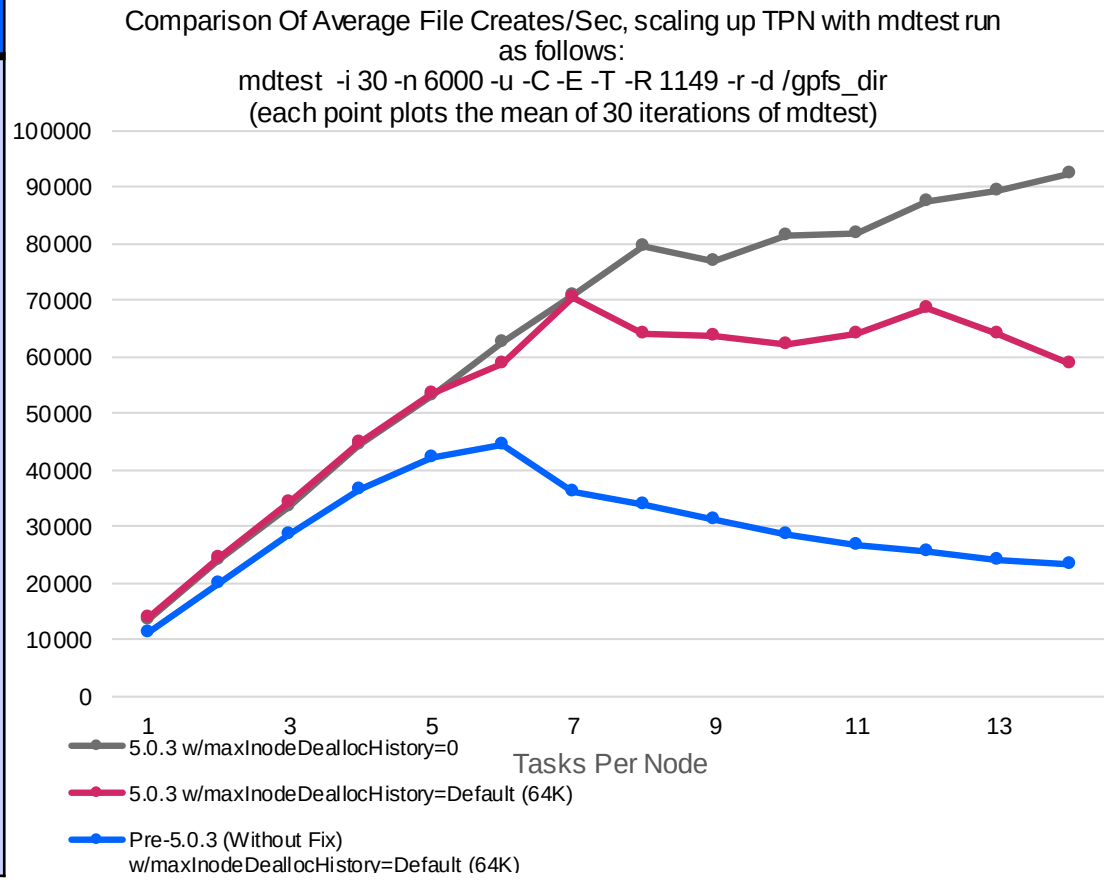
Prior to Spectrum Scale 5.0.3, there's a mutex contention problem (with the SGInodeMapMutex) that impacts Spectrum Scale file create performance as the number of threads creating files on a given node increases, e.g., scaling up MPI tasks per node running mdtest as follows:

mdtest -i $iterations -n $no_files -u -C -E -T –R 1149 -r -d $OUTPUT_DIRECTORY

For file create workloads like the above example, optimal performance can be obtained by applying the fix in 5.0.3 as well as the following two tuning changes:

1. The mmchconfig parameter 'maxFilesToCache' should be set to at least 'no_files' * (number tasks per node)

2. To mimimize additional contention on a second mutex, for workloads that are creating files only (and not concurrently deleting files while creating), the mmchconfig parameter 'maxInodeDeallocHistory' can be reduced or set to 0 (the default value for maxInodeDeallocHistory is the max of 4096 and maxFilesToCache)



mmap Performance Improvements in 5.0.4.3 (1/2)

IBM Spectrum Scale

Locking Issue with mmap read performance description:

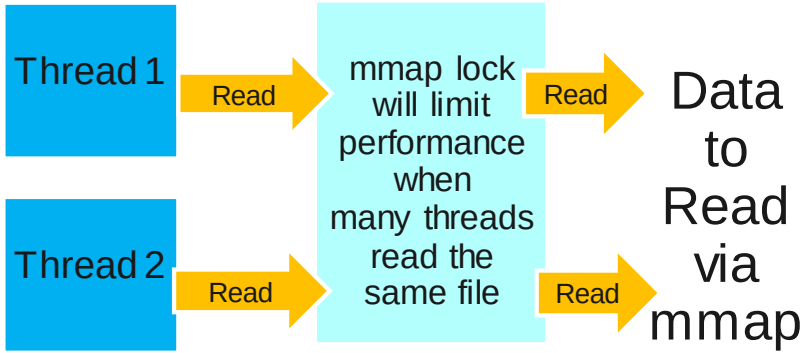
In 5.0.4.3 via APAR IJ22412, we delivered a performance improvement for mmap workloads in which multiple threads/processes read the same file.

This patch set changes the mmapLock to a shared read lock so that all read faults are no longer serialized on the mmapLock and instead a much more efficient read shared lock (MML_READ_SHARED mmap lock) is implemented (serialization is moved to the write path).

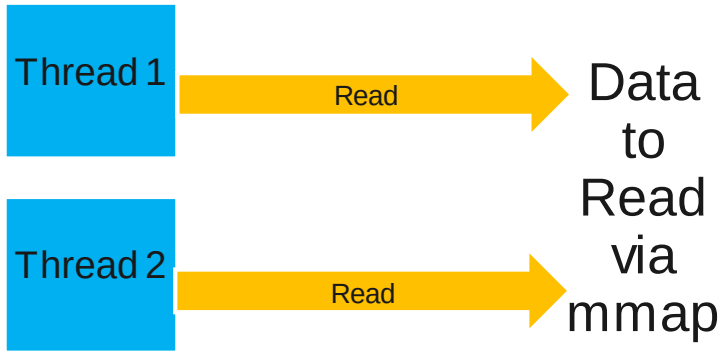
Impact: The change dramatically improves performance when multiple threads/processes are reading from the same file.

Next Steps: improve how prefetching works with multiple threads accessing the same file. Currently prefetching can result in contention around VinfoLock.

Original Flow Prior to 5.0.4



New Flow Prior after 5.0.4



mmap Performance Improvements in 5.0.4.3 (2/2)

IBM Spectrum Scale

Locking Issue with mmap read performance Demonstrating improvement with fio:

To demonstrate the best possible performance improvement resulting from the mmap-related changes in 5.0.4.3 with fio, there are a few steps required:

1. Disable pre-fetch:
mmchconfig prefetchAggressivenessRead=0 -i

(This does not require a restart of GPFS to take effect)

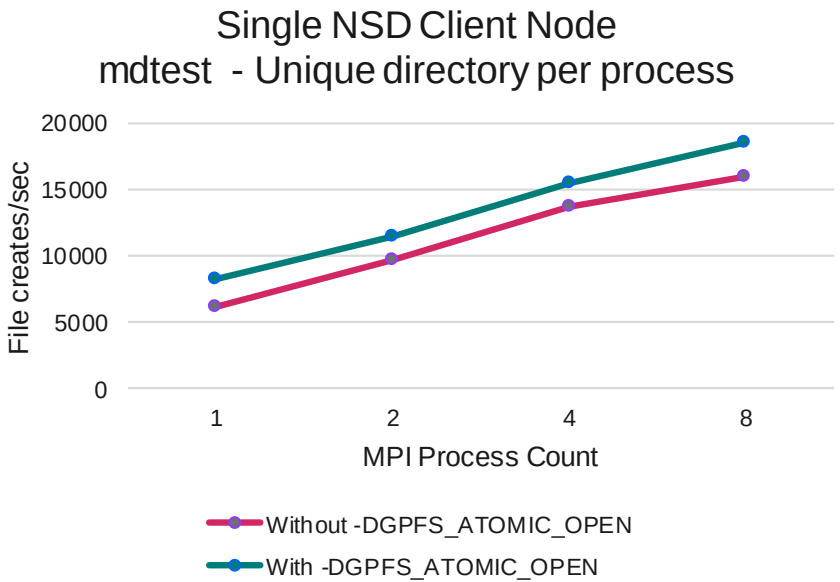
2. There is additional locking associated with the mapping of the address space. It's best to map a single region once and then operate on it (via read/write operations). To minimize the overhead of gpfs_mmap calls in fio, set a large blocksize parameter.

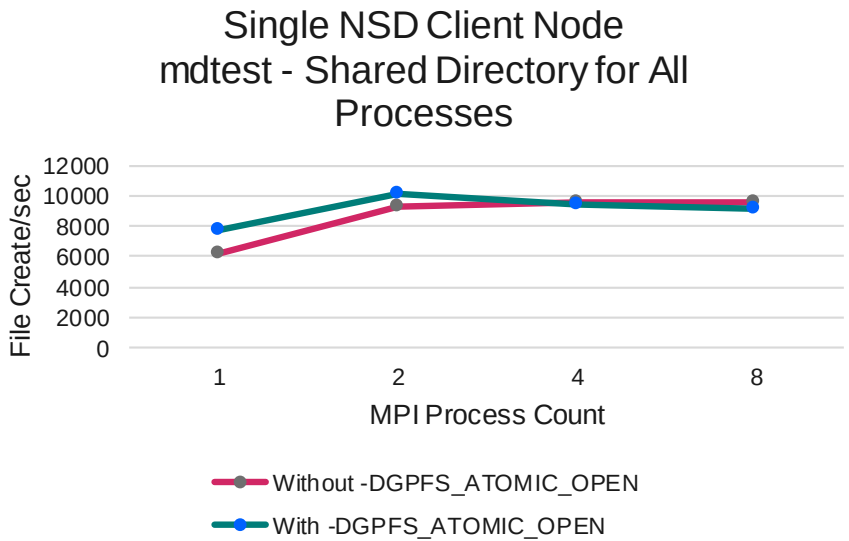
Support for ATOMIC_OPEN on Linux to speed up File create rates

IBM Spectrum Scale

Description of ATOMIC_OPEN enhancement:

- In a future update, Spectrum Scale will deliver support for the ATOMIC_OPEN operation, which will be invoked in cases in which open() with O_CREAT, if the file does *not* exist, Linux (provided the kernel is 3.10 or higher) will call gpfs_i_atomic_open
- The ATOMIC_OPEN path in Spectrum Scale will avoid one round trip call into the kernel, leveraging support for the Linux ATOMIC_OPEN (this optimized path that go to the Spectrum Scale mmfsd daemon to create to the target file).





Performance of File Creates in a Shared Directory

In Dec. 2018 on 4096 nodes, we have data showing file creates, with 1 file per node in a newly created shared directory, typically took between 300--400 seconds for one of our Coral customers (LLNL). The following results come from work that IBM and LLNL did to improve the performance of this file create use case.

Round1 of fixes:

- Improved the efficiency of metanode take-over

- Improved the rate at which directories escape data in inode (base of 5.0.3)

- Performance for 4096 files created in a shared directory: **60–160 seconds**

Round 2 of fixes:

- Added more flexible controls for expanding directory blocks (5.0.3.1-- IJ15858)

- Performance for 4096 files created in a shared directory: **typically ~15 seconds**

Round 3 of fixes:

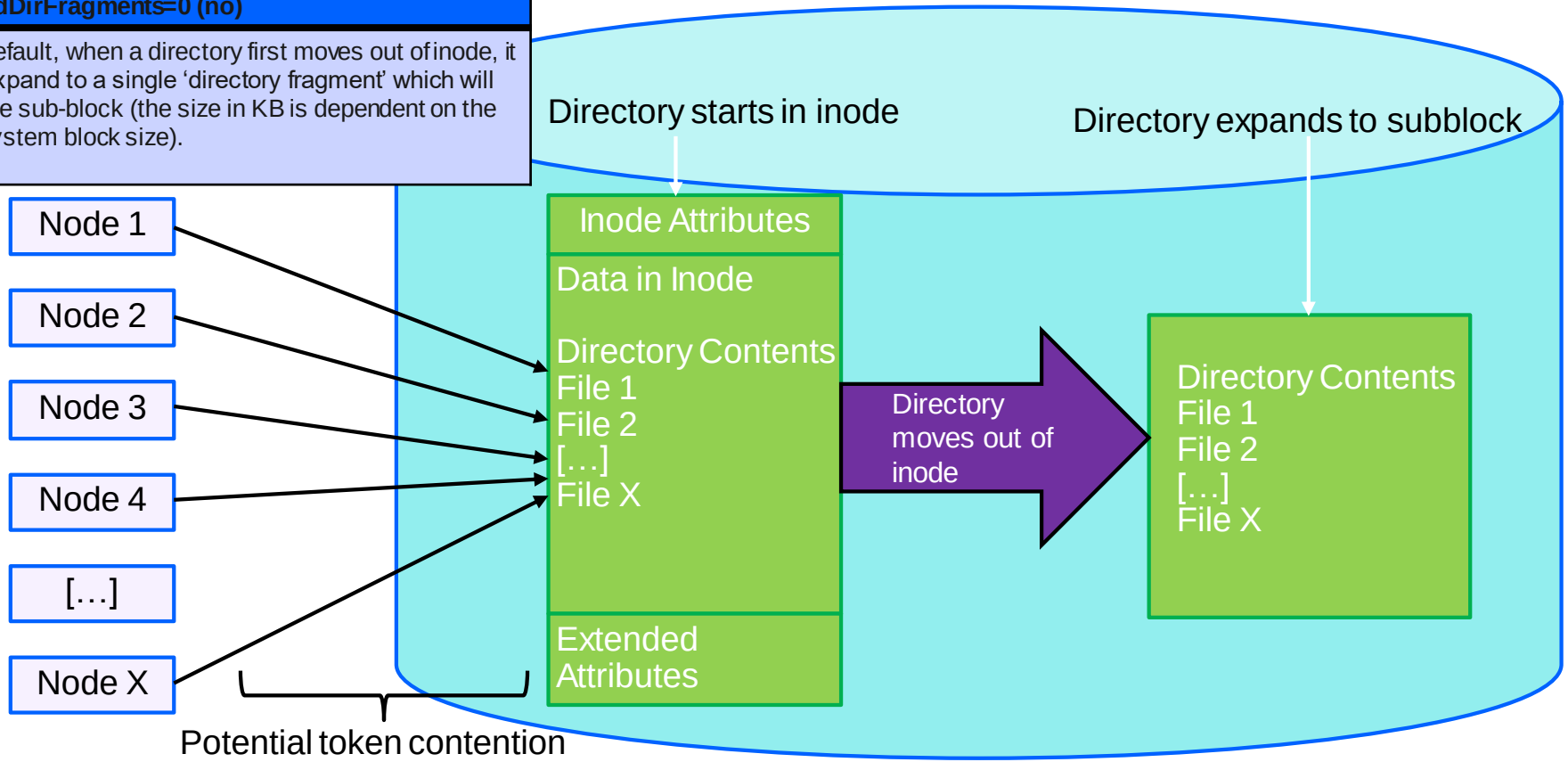
- Added the ability to establish all-to-all daemon connections (slated for a future release)

- Performance for 4096 files created in a shared directory: **typically ~7 seconds**

How a Directory first Expands with the Default Tuning Settings

With Default Tuning Settings
AvoidDirFragments=0 (no)

By Default, when a directory first moves out of inode, it will expand to a single 'directory fragment' which will be one sub-block (the size in KB is dependent on the file system block size).

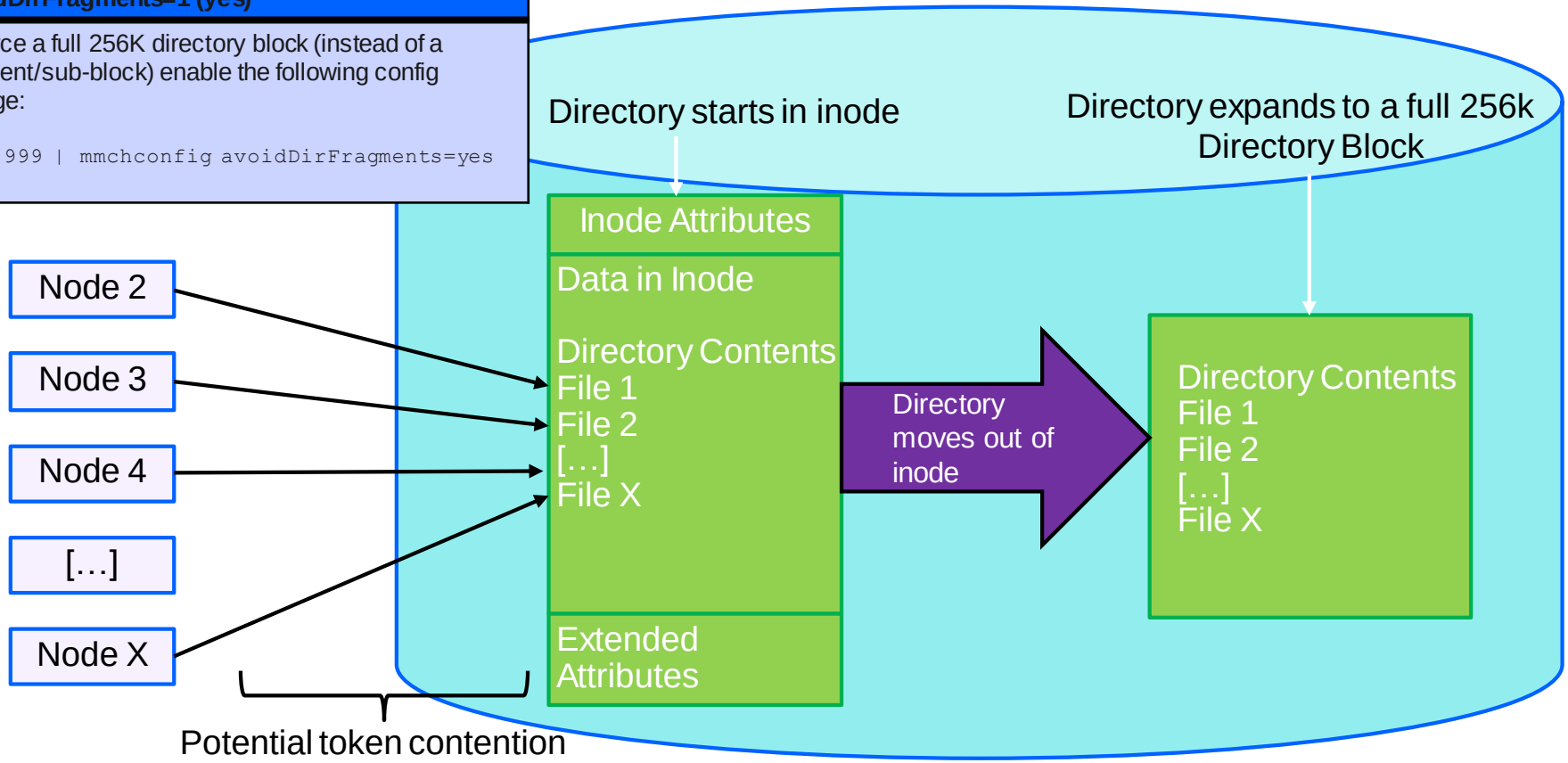


How a Directory first Expands when avoidDirFragments=yes

With Default Tuning Settings
AvoidDirFragments=1 (yes)

To force a full 256K directory block (instead of a fragment/sub-block) enable the following config change:

```
echo 999 | mmchconfig avoidDirFragments=yes
```



Prior to Spectrum Scale 5.0.3, there is Token Contention Moving the Directory out of Inode (1/2)

In cases in which many nodes create a file in a shared directory, the directory will start out as a data in inode mode, but when the data in inode portion of the inode fills, the directory must be expanded out of inode

Multiple nodes will send inode (xw) token requests to move the directory out of data-in-inode

The token manager will grant a token (xw) to the first node that requests this token

The first node changes the inode after acquiring the xw token, ejecting the directory to a subblock (if avoidDirFragments=no)

The second node with a pending xw token request gets a copyset from the TM for nodes with tokens conflicting with the xw token it is requesting

The second node sends requests to the nodes in the copyset asking them to give up their tokens, i.e., to revoke the conflicting tokens

Prior to Spectrum Scale 5.0.3, there is Token Contention Moving the Directory out of Inode (2/2)

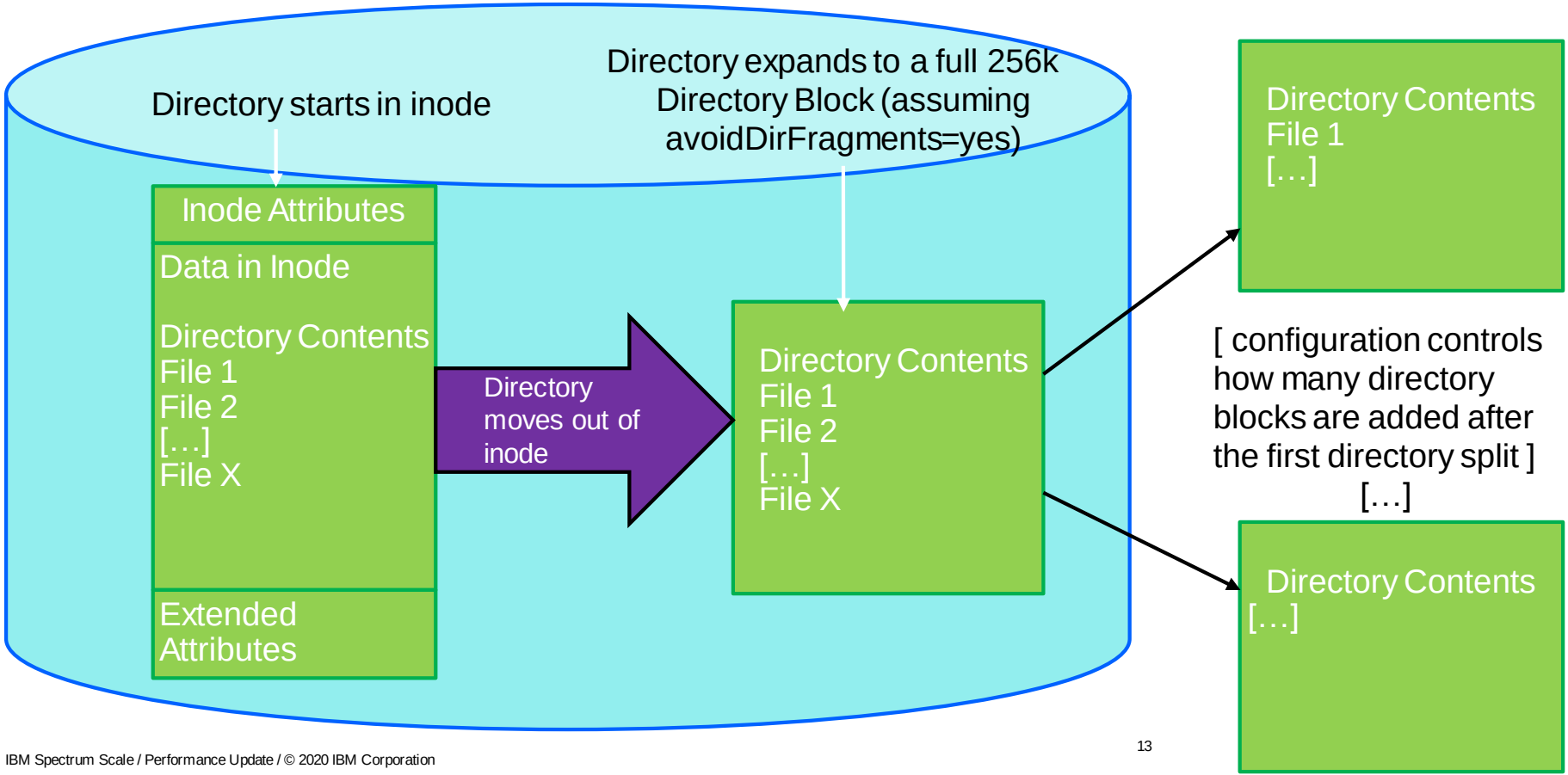
After all conflicting tokens are revoked, the second node sends in the second tell token request to the TM so that the TM knows all conflicting tokens have been released and now the second node can be granted the xw token it wants

The second node, after getting the xw token, checks and realizes that it no longer needs the xw token, because the directory is no longer data-in-inode

The same goes for the third and the remaining pending xw token requests; one by one, they go through the lengthy process to get the xw token, only to find that the inode change has already occurred and so they no longer need the requested xw token to create the file

In 5.0.3, we add a new flag (**CTM_A_XW_FOR_DATA_IN_INODE**) such that we can distinguish cases in which a xw token request is made to request moving the directory out of inode. In such flows, the token manager, on determining that the directory has moved out of inode already, will downgrade the token request to an wf inode token request, and multiple such wf tokens can be granted in parallel for the directory to be written to. This removes the need for all the token revokes that would otherwise occur, starting with the second node that contacts the token manager node to expand the directory out of inode via an xw token. This is the major improvement in 'round 1' of fixes that improved creates on 4096 nodes from 300-400 seconds to 60-160 seconds.

For Coral we added an Undocumented Option to Expand Directories Once Directory Blocks Start to Split (examples shows avoidDirFragments=yes)



Round 2 - Investigating the Performance Overhead of Directory Block Splits (1/2)

In Spectrum Scale 5.0.2.1 (via APAR IJ09151) we added the ability to control how many directory blocks we add to a directory after the first (256K) directory block is split as per the previous chart

However on further analysis of the time it takes to expand directory blocks we found that, to achieve optimal performance, we had to expand the number of blocks that are allocated as soon as the directory escapes the inode (we can't afford to wait for the first block split to occur).

Working with the customer, we started with some experiments using mmchattr (with the -compact option) to expand newly created directories to varying number of blocks:

Directory Size	File Creation Time In Seconds
256KB	143
1 MB	54
4 MB	~5
8 MB	0.2

Note the above measurements were a debug step – they completely remove the overhead of expanding the directory out of inode and also allocating a given number of directory blocks.

These measurements led to adding a new more configurable way of controlling how directory blocks are expanded through the undocumented mmchconfig variable: **DirPreSplitLevels**

Round 2 - Investigating the Performance Overhead of Directory Block Splits (2/2)

- The **DirPreSplitLevels** directory block expansion controls were delivered in Spectrum Scale 5.0.2.1 via APAR IJ09151.

- The new arrangement allows specifying a list of acceptable hash tree levels. Unspecified tree levels are skipped by splitting those blocks, and their children, as needed. In particular, level zero can be skipped, so that once the fragments (less than a full block) are filled the pre-splitting can start immediately. (Setting avoidDirFragments=yes will skip the allocation of fragments.)

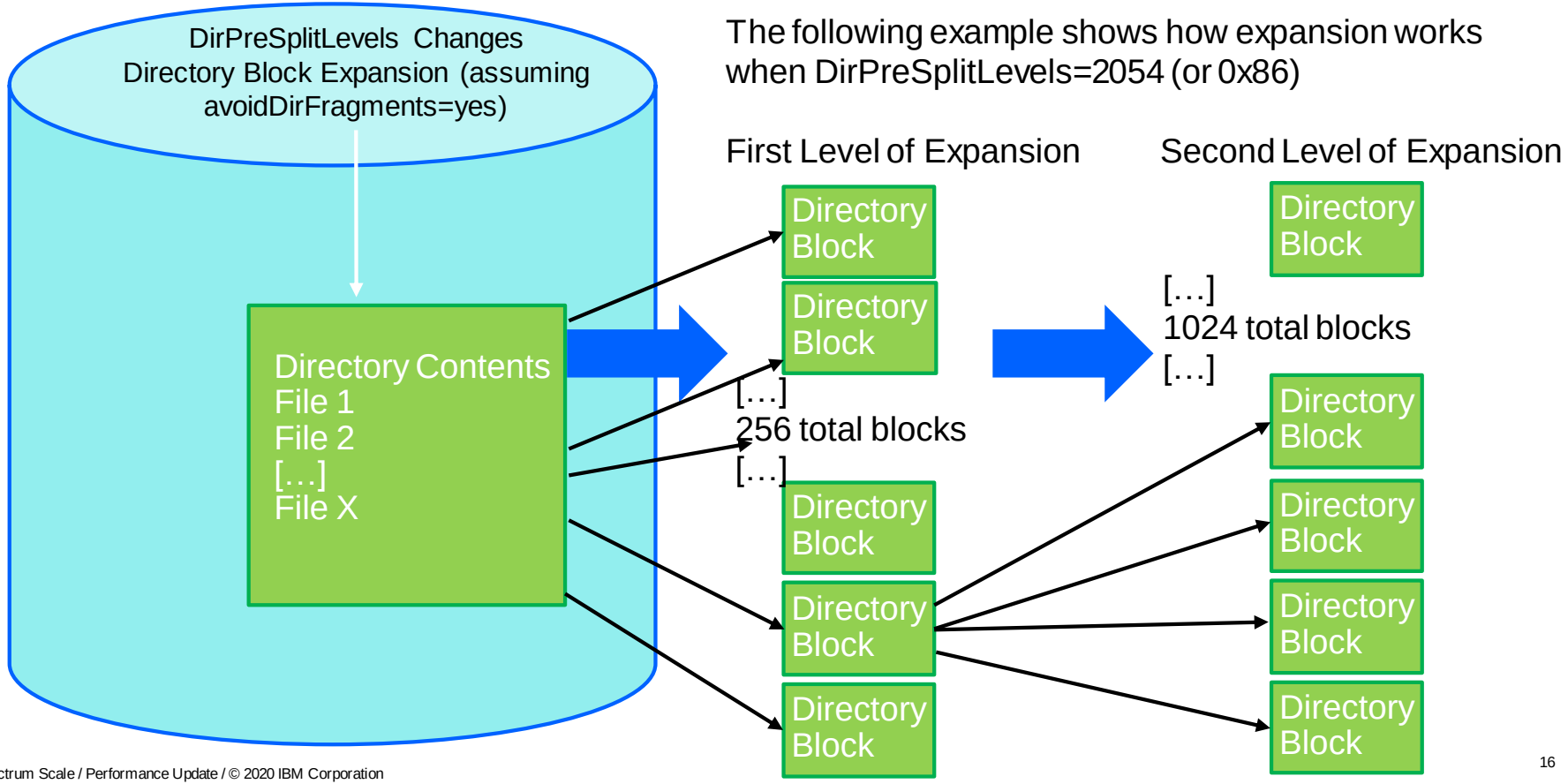
- The hash tree levels (HTL) are encoded as 8-bit values (unsigned char) packed into a 64-bit configuration variable, DirPreSplitLevels. A value of zero provides the default behavior and does no pre-splitting.

- The first level is specified by (DirPreSplitLevels & 256), the second level in the next 8 bits, etc

Value of Byte/Hash Tree Level	Number of Directory Blocks	Total Size of Directory
1	2	512K
2	4	1M
3	8	2M
4	16	4M
5	32	8M
6	64	16M

With DirPreSplitLevels=6 Benchmark times for 4096 node file creates in a new directory improved from 60-160 seconds to typically ~15 seconds.

Example of How Directory Blocks Start to Split when avoidDirFragments=yes And DirPreSplitLevels is Configured (set to 2054 in this Example)



Round 3 – Connecting All of the mmfsd's at Startup/Cluster Join

Based on trace analysis of the case of over 700 nodes creating a file in a newly created directory, we found significant overhead in the initial token revoke that occurs in getting an xw token to expand the directory out of inode

It was one of our lead research developers that first noted that much of the overhead in the token flow was occurring as a result of having to establish a connection from one mmfsd (Spectrum Scale daemon) to all other daemons involved in the file create operation.

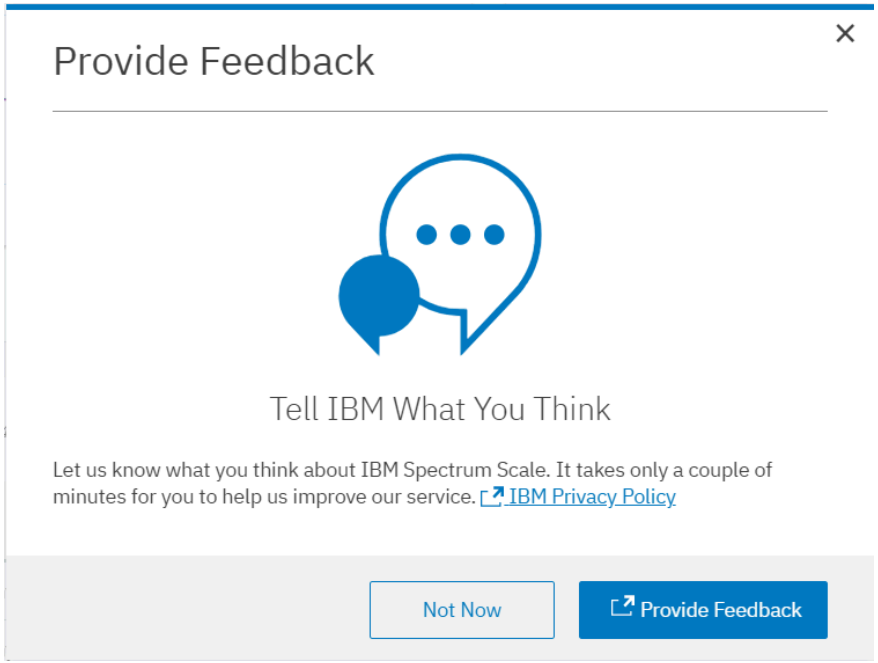
We first did some experiments in which we manually forced all the Spectrum Scale daemons (mmfsd's) to establish all to all connections. These experiments roughly doubled the performance of our previous file creates on 4096 nodes (again for the case of a newly created shared directory)

We implemented new undocumented variables that allow us to control if Spectrum Scale will, at cluster join time, automatically connect to all mmfsd's in a given cluster:

```
allToAllConnection=local     # (means to connect to all other mmfsd's in local cluster)
allToAllConnection=remote    # (means to connect to all mmfsd's on joining a remote cluster)
allToAllConnection=all       # means to connect to both local and remote cluster mmfsd's
```

By setting allToAllConnection=remote we were able to improve typical file creates on 4096 nodes (in a newly created shared directory) from ~15 seconds to ~7 seconds

Thank you!



Please help us to improve Spectrum Scale with your feedback

- If you get a survey in email or a popup from the GUI, please respond
- We read every single reply